

Ressourcen schonen - Datenbanken besser öffnen und schließen

Viele Programme und insbesondere Asp.NET Seiten verwenden Datenbanken, um Daten zu speichern und zu verwalten. Grundsätzlich sollte jeder Entwickler sorgsam mit Verbindungen zu solchen Ressourcen umgehen, d.h. geöffnete Verbindungen müssen immer geschlossen werden sobald diese nicht mehr benötigt werden. Allerdings ist oft genau das Gegenteil der Fall. Dieser Artikel soll kurz aufzeigen, welche Fehler am häufigsten gemacht werden, und wie man mit einem kleinen Trick diese umgehen kann.

Verbindungen zu solchen Ressourcen wie Datenbanken herzustellen benötigt Zeit. Deshalb arbeitet das .NET Framework mit dem sog. *Connection-Pooling*. Der *Connection-Pool* hält eine gewissen Anzahl von möglichen Verbindungen vor und teilt diese den anfragenden Teilen des Programms zu. Dabei wird natürlich darauf geachtet, daß Verbindungen nicht mehr geöffnet sind, bevor sie neu zugeteilt werden. Sobald alle Verbindungen aufgebraucht sind wird eine entsprechende Exception ausgegeben.

Unter normalen Umständen reicht die Anzahl der möglichen Verbindungen aus. Sollte es doch mal zu einem Engpaß kommen ist oft die Unachtsamkeit des Entwicklers schuld, der eine Verbindung nicht geschlossen hat. Aber selbst wenn dieser alles richtig gemacht hat, können offene Verbindungen zurückbleiben.

Häufig werden Datenbanken nach folgendem Schema geöffnet und geschlossen:

```
string sql = "SELECT * FROM Tabelle1";

OdbcConnection sqlConn = new OdbcConnection();
OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

sqlConn.Open();
OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
while (sqlReader.Read())
{
    Console.WriteLine(sqlReader["ID"]);
}

sqlReader.Close();
sqlConn.Close();
```

Dieser Code erfüllt seinen Zweck: Verbindung zur Datenbank wird geöffnet, Daten werden gelesen, Verbindung zur Datenbank wird geschlossen. Was aber passiert mit der Verbindung, wenn während des Lesens der Daten eine Exception auftritt? Sie bleibt geöffnet, zumindest solange bis der *GarbageCollector* die Verbindung irgendwann schließt. Tritt dieser Fehler nun öfters auf, verringert sich die Anzahl der möglichen Verbindungen im *Connection-Pool*, bis eine Exception ausgegeben wird.

Häufig wird jetzt ein *Try-Catch-Finally* Konstrukt in die Methode eingebaut:

```
OdbcConnection sqlConn = new OdbcConnection();
OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

try
{
    sqlConn.Open();
    OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
    while (sqlReader.Read())
    {
        Console.WriteLine(sqlReader["ID"]);
    }
    sqlReader.Close();
}
catch (Exception ex)
{
    // Exception verarbeiten
}
finally
{
    sqlConn.Close();
}
```

Der Code wirkt jedoch unübersichtlich und oft wird im *finally* Abschnitt trotzdem das Schließen der Verbindung vergessen.

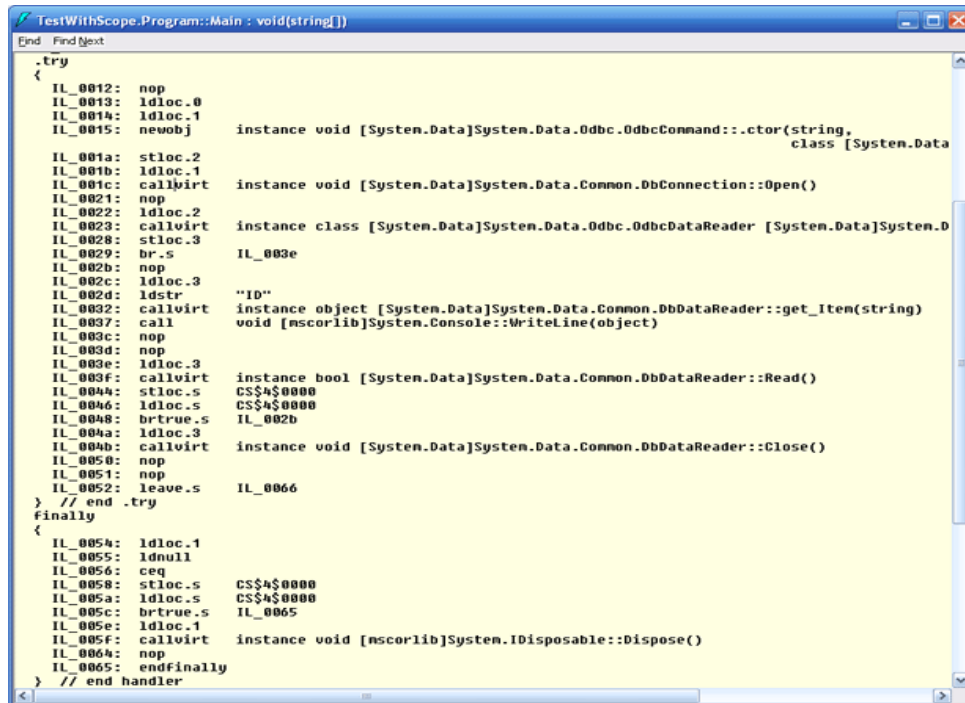
Übersichtlicher Code, der trotzdem im Falle eines Fehler die Verbindung schließt, erhält man mit sog. „*scopes*“. Ein *scope* ist nichts anderes als ein in geschweifte Klammern eingeschlossener Codeabschnitt. Das Objekt, welches die Verbindung zur Datenbank aufbaut, ist also nur innerhalb dieses *scopes* gültig. Im Code würde das folgendermaßen aussehen:

```
using (OdbcConnection sqlConn = new OdbcConnection())
{
    OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

    sqlConn.Open();
    OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
    while (sqlReader.Read())
    {
        Console.WriteLine(sqlReader["ID"]);
    }
    sqlReader.Close();
    sqlConn.Close();
}
```

Diese Vorgehensweise hat mehrere Vorteile. Zum einen wird der Zugriff auf eine bereits geschlossene Verbindung verhindert, da das entsprechende Objekt außerhalb des *scopes* nicht vorhanden ist. Zum anderen wird beim Auftreten eines Fehler in jedem Fall die Verbindung zur Datenbank geschlossen, da der Compiler diesen Code intern in *einen try-catch-finally* Block übersetzt. Somit ist es auch möglich das manuelle Schließen der Datenbank ganz wegzulassen. Der Übersicht halber wird es in diesem Beispiel aber trotzdem verwendet.

Mit Hilfe des Tool `ildasm.exe` kann man dies anhand der vom Compiler erzeugten Metadaten überprüfen:



```

TestWithScope.Program::Main : void(string[])
{
    .try
    {
        IL_0012: nop
        IL_0013: ldloc.0
        IL_0014: ldloc.1
        IL_0015: newobj      instance void [System.Data]System.Data.Odbc.OdbcCommand::.ctor(string,
                                                                    class [System.Data

        IL_001a: stloc.2
        IL_001b: ldloc.1
        IL_001c: callvirt instance void [System.Data]System.Data.Common.DbConnection::Open()
        IL_0021: nop
        IL_0022: ldloc.2
        IL_0023: callvirt instance class [System.Data]System.Data.Odbc.OdbcDataReader [System.Data]System.O
        IL_0028: stloc.3
        IL_0029: br.s      IL_003e
        IL_002b: nop
        IL_002c: ldloc.3
        IL_002d: ldstr    "ID"
        IL_0032: callvirt instance object [System.Data]System.Data.Common.DbDataReader::get_Item(string)
        IL_0037: call     void [mscorlib]System.Console::WriteLine(object)
        IL_003c: nop
        IL_003d: nop
        IL_003e: ldloc.3
        IL_003f: callvirt instance bool [System.Data]System.Data.Common.DbDataReader::Read()
        IL_0044: stloc.5      CS$4$0000
        IL_0046: ldloc.5      CS$4$0000
        IL_0048: brtrue.s   IL_002b
        IL_004a: ldloc.3
        IL_004b: callvirt instance void [System.Data]System.Data.Common.DbDataReader::Close()
        IL_0050: nop
        IL_0051: nop
        IL_0052: leave.s    IL_0066
    } // end .try
    finally
    {
        IL_0054: ldloc.1
        IL_0055: ldnull
        IL_0056: ceq
        IL_0058: stloc.5      CS$4$0000
        IL_005a: ldloc.5      CS$4$0000
        IL_005c: brtrue.s   IL_0065
        IL_005e: ldloc.1
        IL_005f: callvirt instance void [mscorlib]System.IDisposable::Dispose()
        IL_0064: nop
        IL_0065: endfinally
    } // end handler
}

```

Wichtig ist die Zeile:

```
IL_005f:callvirt instance void[mscorlib]System.IDisposable::Dispose()
```

Dieser Befehl gibt die Anweisung, daß alle offenen Ressourcen geschlossen werden sollen.

Mit Hilfe dieser kleinen Änderung am Code kann man nun davon ausgehen, daß die Verbindung zur Datenbank immer geschlossen wird. Außerdem fängt man weitere mögliche Fehler schon beim kompilieren ab.